An Analysis of Performance Enhancement Techniques for Overset Grid Applications NAS-03-008

M.J. Djomehri[*], R. Biswas[†], M. Potsdam[‡], and R.C. Strawn[‡]
NASA Ames Research Center
Moffett Field, CA 94035-1000
{djomehri,rbiswas,potsdam,strawn}@nas.nasa.gov

Abstract

The overset grid methodology has significantly reduced time-to-solution of high-fidelity computational fluid dynamics (CFD) simulations about complex aerospace configurations. The solution process resolves the geometrical complexity of the problem domain by using separately generated but overlapping structured discretization grids that periodically exchange information through interpolation. However, high performance computations of such large-scale realistic applications must be handled efficiently on state-of-the-art parallel supercomputers. This paper analyzes the effects of various performance enhancement techniques on the parallel efficiency of an overset grid Navier-Stokes CFD application running on an SGI Origin2000 machine. Specifically, the role of asynchronous communication, grid splitting, and grid grouping strategies are presented and discussed. Results indicate that performance depends critically on the level of latency hiding and the quality of load balancing across the processors.

1 Introduction

The overset grid methodology [1] for high-fidelity computational fluid dynamics (CFD) simulations about complex aerospace configurations falls into the general class of Schwartz decomposition methods [8]. The solution process resolves the geometrical complexity of the problem domain by generating and using overlapping multi-block structured discretization grids. This overset approach typically employs a Chimera interpolation technique [9] to periodically update and exchange inter-grid boundary information.

However, to reduce time-to-solution, high performance computations of such large-scale realistic applications must be handled efficiently on state-of-the-art parallel supercomputers. Over the years, various parallel programming paradigms have been developed for both distributed and distributed-shared memory systems. Widely used scientific programs suitable for most modern architectures are implemented using a message passing paradigm, such as MPI. Fortunately, the overset grid method can readily employ MPI to exploit parallelism as well as communicate information between distributed overlapping grids.

The parallel efficiency of the overset approach, however, depends upon the proper distribution of the computational workload and the communication overhead among the processors. For a large class of practical problems, optimal load-balancing to minimize processor idle time is a challenging task. Applications with tens of millions of grid points may consist of many overlapping grids. Smart clustering of individual

[*]Employee of Computer Sciences Corporation
[†]NASA Advanced Supercomputing Division
[‡]US Army Aeroflightdynamics Directorate (AMCOM)

grids (also known as blocks or zones) into groups should therefore not only consider the total number of "weighted" grid points, but also the size and connectivity of the inter-grid data. Major challenges during the grouping process may arise due to the wide variation in block sizes and the disparity in the number of inter-grid boundary points. Note also that for large processor sets, the overhead associated with boundary data exchange may adversely affect parallel performance.

This paper analyzes the effects of various performance enhancement techniques on the parallel efficiency of an overset grid Navier-Stokes CFD application called OVERFLOW. Specifically, the role of asynchronous communication, grid splitting, and grid grouping strategies are presented and discussed. First, we study the effect of synchronous and asynchronous communication via MPI. The asynchronous exchange is an attempt to relax the communication schedule in order to hide latency. Second, the splitting of large blocks as a means of controlling the computational load is analyzed. This is particularly important for scalability, where the same grid system must be retained for executing on different numbers of processors. Finally, two grid clustering techniques are examined: one based on a naive bin-packing approach and the other using a more sophisticated graph partitioning method. All our experiments are conducted on an SGI Origin2000 machine using test cases that simulate complex rotorcraft vortex dynamics and consist of more than 63 million grid points. Results indicate that performance depends critically on the level of latency hiding and the quality of load balancing across the processors.

The remainder of this paper is organized as follows. Section 2 provides a brief description of the OVERFLOW application. The performance enhancement techniques of grid splitting, asynchronous communication, and grid grouping are described in Section 3. Parallel performance results are presented and critically analyzed in Section 4. Finally, Section 5 concludes the paper with a summary and some key observations.

2 Numerical Methodology

In this section, we provide a brief overview of the overset grid CFD application called OVERFLOW, including the basics of its solution process, grid connectivity, and message-passing parallelization model.

2.1 Solution Process

The high-fidelity overset grid application, called OVERFLOW [1], owes its popularity within the aerodynamics community due to its ability to handle complex designs consisting of multiple geometric components, where individual body-fitted grids can be constructed easily about each component. The grids are either attached to the aerodynamics configuration (near-body), or are detached (off-body). The union of near- and off-body grids covers the entire computational domain (see Fig. 1(a)).

OVERFLOW uses a Reynolds-averaged Navier-Stokes solver, augmented with a number of turbulence models. In this work, a special version of the code, named OVERFLOW-D [5, 6], is used. Unlike the original version which is primarily meant for fixed-body (static) grid systems, OVERFLOW-D is explicitly designed to simplify the modeling of components in relative motion (dynamic grid systems). For example, in typical rotary-wing problems, the near-field is modeled with one or more grids around the moving rotor blades. The code then automatically generates Cartesian "background" or "wake" grids, called bricks, that encompass these curvilinear near-body grids. At each time step, the flowfield equations are solved independently on each zone in a sequential manner. Overlapping boundary points or inter-grid data are updated from previous solutions prior to the start of the current time step using a Chimera interpolation procedure [9]. The code uses finite differences in space, with a variety of spatial differencing and implicit/explicit temporal time-stepping.

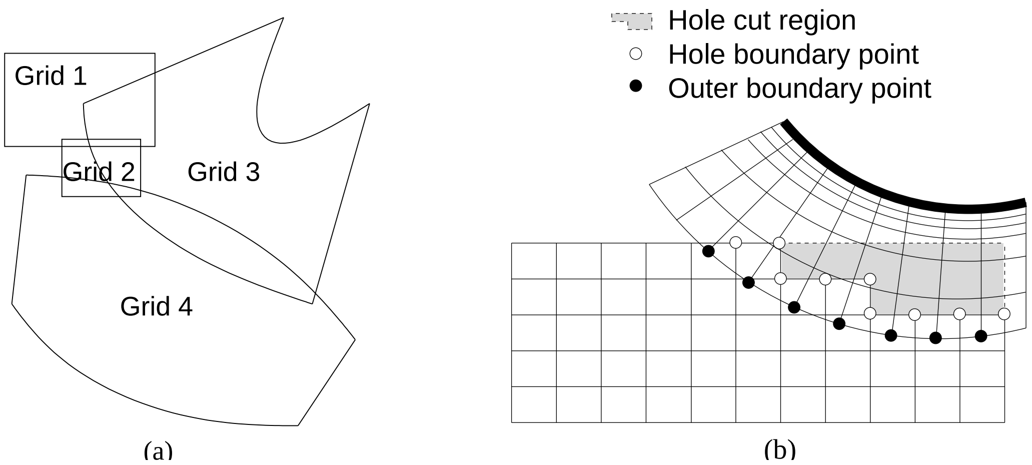


Figure 1: (a) Overset grid schematic; (b) hole and outer inter-grid boundary points.

2.2 Grid Connectivity

The Chimera interpolation procedure [9] determines the proper connectivity of the individual grids. To update inter-grid boundary data, the scheme has to process two types of boundary points: “hole” and “outer” boundary points (see Fig. 1(b)). Holes are cut in grids which intersect solid surfaces, such as when a portion of an overset grid lies inside a physical body. The hole boundary points are on the surfaces of these cuts. All other inter-grid boundary points are classifi ed as outer. Adjacent grids are expected to have at least a one-cell, or a single fringe, overlap to ensure the continuity of the solutions; for higher-order accuracy and to retain certain physical features in the solution, a double fringe overlap is sometimes sought [10]. A program named Domain Connectivity Function (DCF) [7] computes the inter-grid donor points that have to be supplied to other grids. The DCF procedure is incorporated into the OVERFLOW-D code and fully coupled with the flow solver. For dynamic grid systems, DCF has to be invoked at every time step to create new holes and inter-grid boundary data.

2.3 MPI Parallelization Model

The parallel version of the OVERFLOW-D application has been developed around the multi-block feature of the sequential code, which offers a natural coarse-grained parallelism [13]. The main computational logic at the top level of the sequential code consists of a “time-loop”, a “grid-loop”, and a “subiteration-loop”. The last two loops are nested within the time-loop. Within the grid-loop, solutions are obtained on the individual grids with imposed boundary conditions, where the Chimera interpolation procedure successively updates inter-grid boundaries after computing the numerical solution on each grid. Convergence of the solution process is accelerated by the subiteration-loop. Upon completion of the grid-loop, the solution is automatically advanced to the next time step by the time-loop. The overall procedure may be thought of as a Gauss-Seidel iteration.

A message passing programming model based on the MPI library was implemented using the single program multiple data (SPMD) paradigm. To facilitate parallel execution, a grouping strategy is required to assign each grid to an MPI process. The total number of groups, G, is equal to the total number of MPI processes, P. Since a grid can only belong in one group, the total number of grids, Z, must be at least equal to P. If Z is larger than P, a group will consist of more than one grid. Two techniques for clustering grids into groups is discussed later in Section 3.3.

The logic in the MPI programming model differs slightly from that of the sequential case ($G = P = 1$). Here the grid-loop is subdivided into two procedures, a loop over groups (“group-loop”) and a loop over

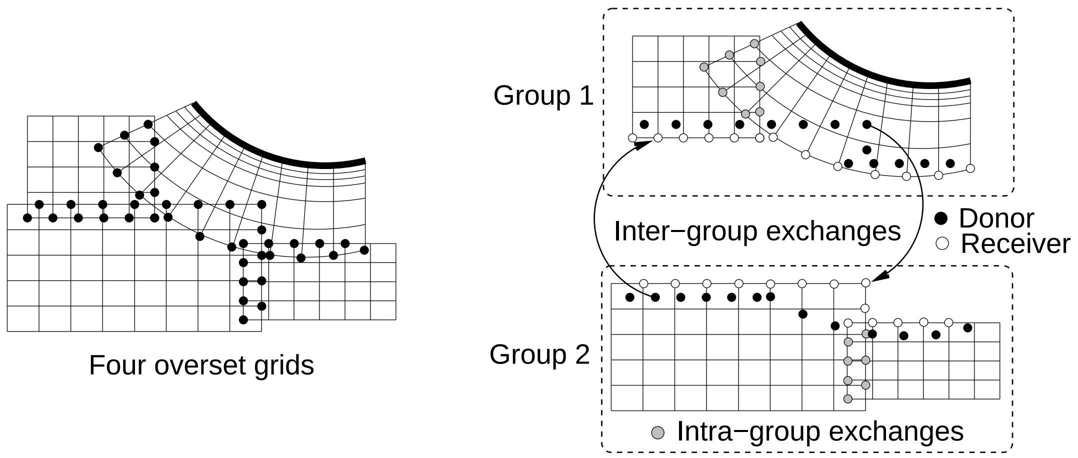


Figure 2: Overset grid intra-group and inter-group communication.

the grids within each group. Since each MPI process is assigned to only one group, the group-loop is performed in parallel, with each group performing its own sequential grid-loop. The inter-grid boundary updates among the grids within each group (these are also called intra-group updates) are performed as in the serial case. Chimera updates are also necessary for overlapping grids that are in different groups, and are known as inter-group exchanges (see Fig. 2). The inter-group donor points from grids in group G_i to grids in group G_j are stored in a send buffer and exchanged between the corresponding processes via MPI calls. These inter-group exchanges are transmitted at the beginning of every time step based on the interpolatory updates from the previous time step.

3 Performance Enhancement Techniques

We have developed and utilized various performance enhancement techniques to improve the parallel efficiency of the OVERFLOW-D application. Specifically, the role of asynchronous communication, grid splitting, and grid grouping strategies are presented and discussed in this section. Superior parallel performance of such large-scale realistic applications on state-of-the-art commercial supercomputers is critical to advance our scientific understanding and problem solving capability.

3.1 Asynchronous Communication

Almost all of the communication that is required in the OVERFLOW-D application concerns the exchange of inter-grid boundary data, and is contained in the subroutine, `qbc_exchange`. The message passing can be synchronous or asynchronous, but the choice significantly affects the MPI programming model. The synchronous communication can be performed with blocking MPI send/receive calls, while the asynchronous communication uses non-blocking calls.

With synchronous communication, the total number of send/receive calls is $P \times (P-1)$, counting even the messages of zero length. The send calls complete only when the receiving processor is ready to accept the messages, i.e., the matching receive calls are posted. The increase in execution time caused by this communication pattern is analogous to the introduction of an implicit serialization into the code. The initial parallel version of OVERFLOW-D was implemented with synchronous message passing and tested with a relatively small dataset on 16 processors [13]. As a result, the communication time was quite insignificant and therefore acceptable. However, performance analysis using a larger dataset and

```
/* Send data from group $ND$ to group $NR$ */
do $ND = 1,\ G$
    if (myrank .eq. $ND$) then
        do $NR = 1,\ G$
            MLEN_SEND = ISNT ($NR$)                 /* Set length of send array */
            if (myrank .ne. $NR$) then
                call MPI_SEND (QBCSND, MLEN_SEND, ···)
            else
                do $I = 1,$ MLEN_SEND
                    QBCRCV ($I$) = QBCSND ($I$)     /* Memory copies */
                end do
            end if
        end do
    else
/* Receive data from group $ND$ */
        MLEN_RECV = IRCV ($ND$)                     /* Set length of receive array */
        call MPI_RECV (QBCRCV, MLEN_RECV, ···)
    end if
end do
```

Figure 3: Outline of the synchronous communication model in the original OVERFLOW-D code.

more processors (presented in Section 4) indicate a serious communication bottleneck for the exchange of boundary data via the synchronous approach.

In order to be better able to compare the original synchronous and our new asynchronous communication strategies, we present in Fig. 3 an outline of the synchronous model. The group boundary data arrays are specified by QBCSND and QBCRCV with total lengths of MLEN_SEND and MLEN_RECV, respectively. These values are determined for each group (processor) by pertinent arrays, ISNT and IRCV, each of length G.

Our first performance enhancement technique is to use asynchronous communication for inter-grid boundary data exchange within subroutine qbc_exchange. The asynchronous strategy is an attempt to relax the communication schedule in order to hide latency. Asynchronous communication consists of non-blocking send/receive calls. Unlike the corresponding blocking calls, these invocations place no constraints on each other in terms of completion. Receive completes immediately, even if no messages are available, and hence allows maximal concurrency. The non-blocking receives are posted by receiving processors prior to the pertinent sends from the sending processors. Furthermore, messages of zero length are not sent to decrease the communication overhead. We have implemented this asynchronous message passing model in the current version of OVERFLOW-D.

In general, however, control flow and debugging can become a serious problem if, for instance, the order of messages needs to be preserved. Fortunately, in the overset grid application, the Chimera boundary updates take place at the completion of each time step, and the computations are independent of the order in which messages are sent or received. Being able to exploit this fact allows us to easily use asynchronous communication within OVERFLOW-D. Figure 4 gives an outline of the asynchronous approach that we have implemented. The same naming convention discussed with respect to the synchronous case is also adopted here.

```
/* Post receives in group $NR$ from group $ND$ */
do $ND = 1,\ G$
    MLEN_RECV = IRCV ($ND$)                      /* Set length of receive array */
    if (MLEN_RECV .ne. 0) then
        if (myrank .ne. $ND$) then
            call MPI_IRECV (QBCRCV, MLEN_RECV, $\cdots$)
        end if
    end if
end do
/* Send data from group $ND$ to group $NR$ */
do $NR = 1,\ G$
    MLEN_SEND = ISNT ($NR$)                      /* Set length of send array */
    if (myrank .ne. $NR$) then
        if (MLEN_SEND .ne. 0) then
            call MPI_ISEND (QBCSND, MLEN_SEND, $\cdots$)
        end if
    else
        do $I = 1,$ MLEN_SEND
            QBCRCV ($I$) = QBCSND ($I$)          /* Memory copies */
        end do
    end if
end do
/* Check that all receives have completed */
call MPI_WAITALL
```

Figure 4: Outline of our asynchronous communication model in OVERFLOW-D.

3.2 Grid Splitting

Load balancing is critically important for efficient parallel computing. The objective is to distribute equal computational workloads among the processors while minimizing the inter-processor communication cost. On a given platform, the primary procedure that affects the load balancing of an overset grid application is the grid grouping strategy. To facilitate parallel execution, each grid must be assigned to an MPI process. Since the total number of grids, Z, is at least equal to the number of processes, P, a proper clustering of the grids into G groups is required ($G = P$).

Unfortunately, the size of the Z blocks in an overset grid system may vary substantially, thereby complicating the grouping procedure and significantly affecting the overall load balance. For instance, each near-body block is a three-dimensional curvilinear structured grid generated about the geometric components of an aerodynamics configuration. The ordered triplet, (i, j, k), represents the three spatial dimensions and varies from (1,1,1) to a maximum of (I, J, K), for a block with $I \times J \times K$ grid points. The dimensions of each block are primarily selected to introduce proper refinement into the grid spacing in an effort to maintain certain features of the physical solution, but have no bearing on the type of computations used, serial or parallel. Consequently, there may be orders of magnitude differences in near-body block sizes for the initial grid system. Recall that these near-body grids overlap the Cartesian wake (off-body) grid system to cover the entire computational domain.

A smart mechanism is therefore needed to limit the size of the individual blocks. One option is to add some control during the grid generation process, but this would further complicate an already complex task.

The strategy would also require information about the number of groups (G) which may vary from one simulation run to the next depending on the chosen number of processors (P), since G must be equal to P. The second approach, which we have implemented as part of this work, is to split the largest blocks into sub-blocks of desired sizes prior to grouping them. This performance enhancement technique is independent of the grid generation procedure and is automatically implemented at runtime, prior to the start of the time-loop.

The original version of OVERFLOW-D has the ability to perform some automatic grid splitting without any user input, but it was only to ensure there were enough blocks Z to form G groups with $G = P$. However, for large test cases such as those used in this paper, further control is required. In particular, we must maintain exactly the same blocks in the grid system when executing on different numbers of processors to examine code scalability. In our latest version of OVERFLOW-D, the user specifi es two input parameters, *maxnb* and *maxgrd*, for splitting the largest blocks based on some knowledge of the initial grid system and the maximum target value of P. All near- and off-body grids larger in size than *maxnb* and *maxgrd*, respectively, are then split into overlapping sub-blocks of smaller but equal sizes. For near-body grids, the split is done in one dimension only, i.e., for one of the indices i, j, or k, depending on the values of I, J, and K. The type of imposed boundary condition (periodic, reflecting, etc.) also plays a role in determining the splitting direction. For the uniform Cartesian off-body grids, splitting can be performed in multiple dimensions if necessary.

Conceptually, having smaller block sizes simplifi es the load balancing procedure and leads to a more computationally balanced workload; however, a limiting drawback is an increase in the ratio of surface-to-volume grid points. A large value of this ratio indicates that the amount of overlap boundary data to be transferred via point-to-point communication between pairs of processors has increased disproportionately relative to the computational workload. Furthermore, since it is necessary to maintain at least a single (one-cell) and sometimes even a double (two-cell) fringe overlap between adjacent blocks, the total number of grid points increases during the splitting process, resulting in a larger computational and communication load per processor.

3.3 Grid Grouping

As mentioned in Section 3.2, the grid grouping strategy has a substantial effect on the quality of load balancing for an overset grid application like OVERFLOW-D. The grouping is a function of the following parameters: execution time per grid point, total number of grid points per block, number of blocks Z, volume of the total boundary data to be exchanged per processor, rate of communication, and total number of processors P. In principle, grouping depends only on the characteristics of the grids and their connectivity; it does not take into account the topology of the physical processors. The assignment of groups to processors is somewhat random, and is taken care of by the system job scheduler usually based on a fi rst-touch strategy at the time of the run.

The original parallel version of OVERFLOW-D uses a grid grouping strategy based on a bin-packing algorithm [13]. It is one of the simplest clustering techniques that strives to maintain a uniform number of “weighted” grid points per group while retaining some degree of connectivity among the grids within each group. Prior to the grouping procedure, each grid is weighted depending on the physics of the solution sought. The goal is to ensure that each weighted grid point requires the same amount of computational work. For instance, the execution time per point belonging to near-body grids requiring viscous solutions is higher than that for the inviscid solutions of off-body grids. The weight can also be deduced from the presence or absence of a turbulence model. The bin-packing algorithm then sorts the grids by size in descending order, and assigns a grid to every empty group. Therefore, at this point, the G largest grids are each in a group by themselves. The remaining grids are then handled one at a time. Each grid is assigned to the smallest group that satisfi es the connectivity test with other grids in that group. The connectivity test only inspects for an overlap between a pair of grids regardless of the size of the boundary data or their connectivity to other neighboring grids. The process terminates when all grids are assigned to groups.

Our third performance enhancement technique is to devise and implement a more sophisticated grid grouping algorithm that incorporates more of the parameters discussed at the beginning of this section. It is based on a graph representation of the overset grid system. The nodes of the graph correspond to the near- and off-body grids, while an edge exists between two nodes if the corresponding grids overlap. The nodes are weighted by the weighted number of grid points and the edges by the communication volume. Given such a graph with Z nodes and E edges, the problem is to divide the nodes into G sets such that the sum of the nodal weights in each set are almost equal while the sum of the cut edges' weights is minimized. This is the classical graph partitioning problem that is widely encountered in the parallel computing arena [12]. Such a grouping strategy is, in theory, more optimal than bin-packing in that the grids in each group enjoy higher intra-group dependencies with fewer inter-group exchanges.

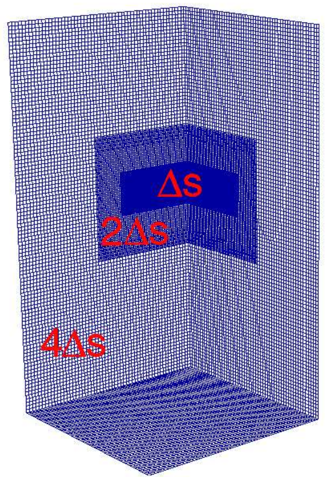
The graph partitioning algorithm that we have implemented in OVERFLOW-D to address the grid grouping problem is based on EVAH [4]. It consists of a set of allocation heuristics that considers the constraints inherent in multi-block CFD problems. EVAH was initially developed to predict the performance scalability of overset grid applications on large numbers of processors, particularly within the context of distributed grid computing across multiple resources [2]. In this work, we have modifi ed EVAH to cluster overset grids into groups while taking into account their relative overlaps.

Among several heuristics that are available within EVAH, we have used the one called *largest task first with available communication costs* (LTF_ACC). In the context of the current work, a task is synonymous with a block in the overset grid system. The size of a task is defi ned as the computation time for the corresponding block. LTF_ACC is constructed from the basic *largest task first* (LTF) heuristic that is identical to the bin-packing strategy. LTF is then enhanced by the systematic integration of the status of the processes in the form of *minimum finish time* and *largest idle time*. Because of the overhead involved due to data exchanges between neighboring zones, the assignment heuristic is further refi ned to integrate communication costs to model their impact on the overall execution time. A procedure has been developed to interface the DCF subroutine of OVERFLOW-D with EVAH heuristics. Extensive details of this approach is available in [4].

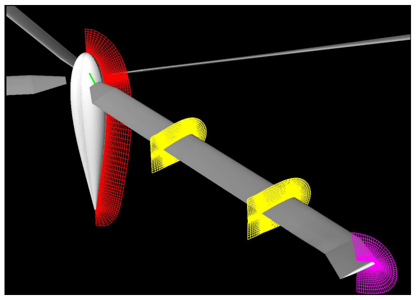
4 Parallel Performance Results

The CFD problem used for the experiments in this paper is a Navier-Stokes simulation of vortex dynamics in the complex wake flow region for hovering rotors. Figure 5 shows sectional views of the test application grid system. The Cartesian off-body wake grids surround the curvilinear near-body grids with uniform resolution, but become gradually coarser upon approaching the outer boundary of the computational domain. Specifi cally, the spacing of the off-body grid nearest the rotor blade is Δs, that for the next surrounding level is $2\Delta s$, and so on for every successive level. Figure 6 shows a cut plane through the computed vortex wake system including vortex sheets as well as a number of individual tip vortices. A complete description of the underlying physics and an extensive analysis of the numerical simulations pertinent to this test problem can be found in [11]. We have used the following three cases to evaluate our performance enhancement techniques discussed in Section 3:

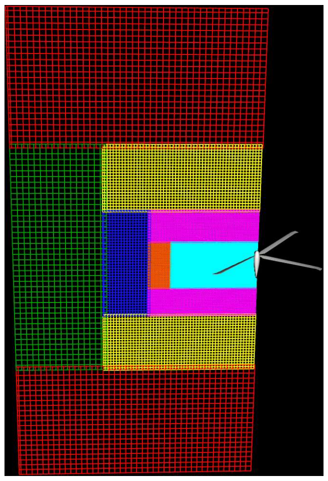
- Case 1: Z = 454, $\sim$63M grid points, *maxnb* = 250K, *maxgrd* = 300K.
- Case 2: Z = 857, $\sim$69M grid points, *maxnb* = 100K, *maxgrd* = 100K.
- Case 3: Z = 1679, $\sim$78M grid points, *maxnb* = 60K, *maxgrd* = 70K.



(a)



(b)



(c)

Figure 5: Sectional views of the test application grid system: (a) off-body Cartesian wake grids, (b) near-body curvilinear grids, and (c) cut plane through the off-body wake grids surrounding the hub and rotors.

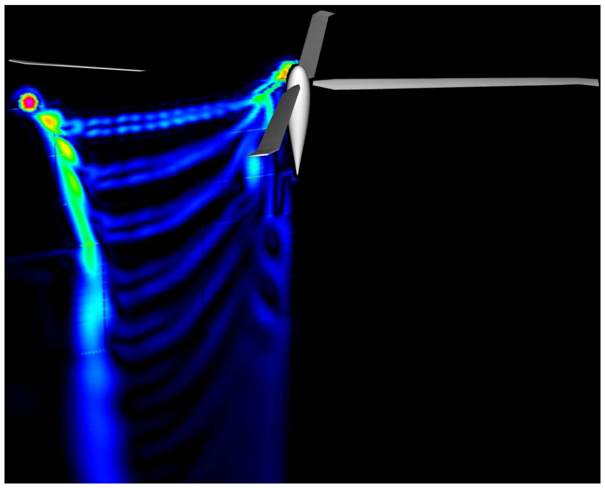


Figure 6: Computed vorticity magnitude contours on a cutting plane located 45^o behind the rotor blade.

All experiments were run on the 512-processor SGI Origin2000 shared-memory system at NASA Ames Research Center. Each Origin2000 node is a symmetric multiprocessor (SMP) containing two 400 MHz MIPS R12000 processors and 512 MB of local memory. Due to the size of the test problem, all runs were conducted for only 100 time steps. Our timing results are averaged over the number of iterations and given in seconds.

4.1 Asynchronous Communication Results

Table 1 shows a comparison of various timings for Case 1 using synchronous (blocking send/receive) and asynchronous (non-blocking) communication. The grouping algorithm is the basic bin-packing strategy that is available with the original version of OVERFLOW-D. The execution time T_{exec} is the average time required to solve every time step of the application, and includes the computation, communication, Chimera interpolation, and processor idle time. The average computation (T_{comp}^{avg}) and communication (T_{comm}^{avg}) times over P processors are also shown. Finally, the maximum computation (T_{comp}^{max}) and communication (T_{comm}^{max}) times are reported and used to measure the quality of load balancing for each run. The computation load balance factor (LB_{comp}) is the ratio of T_{comp}^{max} to T_{comp}^{avg}, while the communication load balance factor (LB_{comm}) is the ratio of T_{comm}^{max} to T_{comm}^{avg}. Obviously, the closer these factors are to unity, the higher is the quality of load balancing.

Table 1: Runtimes (in seconds) and load imbalance factors with synchronous and asynchronous communication, and bin-packing grouping strategy for Case 1

	Synchronous						
P	T_{exec}	T_{comp}^{max}	T_{comp}^{avg}	T_{comm}^{max}	T_{comm}^{avg}	LB_{comp}	LB_{comm}
32	37.7	31.9	24.1	4.4	4.3	1.32	1.02
64	22.1	17.0	12.5	4.3	4.2	1.36	1.02
128	14.0	8.8	6.3	4.3	4.3	1.40	1.00
256	13.0	6.0	3.2	6.4	6.4	1.87	1.00
320	14.8	5.3	2.7	9.2	8.0	1.96	1.15
384	16.6	5.2	2.0	9.9	9.9	2.60	1.00
448	18.3	9.9	1.8	11.5	11.4	5.50	1.01
	Asynchronous						
P	T_{exec}	T_{comp}^{max}	T_{comp}^{avg}	T_{comm}^{max}	T_{comm}^{avg}	LB_{comp}	LB_{comm}
32	34.6	32.7	24.5	0.70	0.61	1.33	1.15
64	18.0	16.8	12.4	0.41	0.35	1.35	1.17
128	9.8	8.8	6.3	0.36	0.31	1.40	1.16
256	7.0	5.9	3.2	0.37	0.30	1.84	1.23
320	6.9	5.1	2.6	0.98	0.68	1.96	1.44
384	6.8	6.0	2.0	0.48	0.36	3.00	1.33
448	7.0	6.3	1.8	0.49	0.43	3.50	1.14

Notice that the computational workload for both runs is identical as is evidenced by the fact that T_{comp}^{avg} is essentially the same. However, T_{exec} for asynchronous communication is consistently lower, and shows bigger improvements as the number of processors increases. In fact, for $P > 256$, the non-blocking communication strategy reduces T_{exec} by more than a factor of two. The reason for this improvement can be found in the communication times. A comparison shows that T_{comm}^{max} and T_{comm}^{avg} for the synchronous runs are an order of magnitude larger than the corresponding times for the asynchronous communication. This is reflected in T_{exec} where communication usually accounts for less that 6% for the asynchronous case, but is more than 50% for many of the synchronous runs.

Scalability for the asynchronous case for $P \leq 256$ is signifi cantly better than its synchronous counterpart. For $P \geq 320$, scalability suffers for both cases not only due to the relatively larger communication overhead but also because of workload imbalance. The latter can be observed from the increasing value of LB_{comp}. However, LB_{comm} shows that communication is well-balanced across all processors for most runs, particularly for blocking communication (which is due to the very nature of synchronous communication). Overall results indicate the general superiority of the non-blocking asynchronous approach over synchronous communication for this application.

4.2 Grid Splitting Results

The impact of grid splitting on load balancing quality is investigated for all the three cases. Case 1 models the situation where the number of splits per block is low, or equivalently, the sizes of the newly-created sub-blocks are quite large. In Case 2, the sizes of the sub-blocks are somewhat smaller, while in Case 3, they are even more so. All runs use asynchronous communication and the bin-packing strategy to cluster grids into groups. Timings for Case 1 are shown in Table 1, while those for the other two cases are presented in Table 2.

Table 2: Runtimes (in seconds) and load imbalance factors with asynchronous communication and bin-packing grouping strategy for Cases 2 and 3

	Case 2						
P	T_{exec}	T_{comp}^{max}	T_{comp}^{avg}	T_{comm}^{max}	T_{comm}^{avg}	LB_{comp}	LB_{comm}
32	32.0	29.3	23.0	1.30	0.90	1.27	1.44
64	13.5	11.9	10.8	0.67	0.55	1.10	1.22
128	7.6	6.2	5.4	0.60	0.52	1.15	1.15
256	5.5	3.7	2.8	0.88	0.50	1.32	1.76
320	4.7	2.9	2.2	0.57	0.46	1.32	1.24
384	4.7	2.9	1.9	0.99	0.56	1.53	1.77
448	4.5	3.0	1.7	0.85	0.46	1.76	1.85
	Case 3						
P	T_{exec}	T_{comp}^{max}	T_{comp}^{avg}	T_{comm}^{max}	T_{comm}^{avg}	LB_{comp}	LB_{comm}
32	39.9	34.5	27.8	3.10	2.20	1.24	1.41
64	13.8	11.4	10.4	0.95	0.75	1.10	1.27
128	7.9	6.4	5.2	0.85	0.70	1.23	1.21
256	4.5	3.1	2.6	0.95	0.68	1.19	1.40
320	4.3	2.8	2.1	0.90	0.61	1.33	1.48
384	4.0	2.4	1.8	0.65	0.57	1.33	1.14
448	3.8	2.3	1.6	0.71	0.60	1.44	1.18

The overall quality of computational workload balancing for the three cases can be observed by comparing LB_{comp} from Tables 1 and 2. Obviously, the factor increases with the number of processors as load balancing becomes more challenging with a fixed problem size. As expected, Case 3 exhibits the best quality for any given value of P because it has the largest number of grids which are all generally smaller, i.e., it has the finest granularity. It should be noted here that though we are evaluating the level of workload imbalance from the runtimes, the grid splitting and grouping strategies are based on the number of weighted grid points. Computed from that perspective, the load balance quality is somewhat better but follows the same trend.

Let us now look at the communication times. Notice that both T_{comm}^{max} and T_{comm}^{avg} generally increase with increasing number of blocks (Case 1 through Case 3). (The communication time also depends on the topology and the connectivity of the grid system.) This is because even though grid splitting has a positive impact on the computational load balance, it adversely affects the communication time. Basically, the surface area increases with the number of blocks, thereby increasing the volume of the boundary exchange data. For example, the ratio of surface-to-volume grid points for the three cases are 11%, 14%, and 18%, respectively. Communication therefore also accounts for a larger percentage of the total execution time. The ratio of T_{comm}^{avg} to T_{exec} is 2–10% for Case 1, 3–12% for Case 2, and 5–16% for Case 3.

Conceptually, the splitting of grids into smaller blocks should also improve the communication load balance LB_{comm}. Clearly, Case 3 does not have the best overall communication load balance, and Case 2 has poorer quality than Case 1. These results indicate that the optimal choice of the splitting parameters *maxnb* and *maxgrd* depends on the number of processors used. However, in our experiments, we wanted a fixed grid system independent of the processor count. We should also note that because of the complex nature of OVERFLOW-D, grid splitting has been implemented in only one coordinate direction at this time for the near-body grids. Even if it were available in multiple directions, most grids would not benefit due to boundary condition and viscous direction splitting restrictions.

Finally, parallel scalability also improves with more blocks. This can be observed by comparing the T_{exec} times in Tables 1 and 2. In fact, we obtain superlinear speedup between 32 and 64 processors for Cases 2 and 3. Also, T_{exec} decreases consistently for Case 3 all the way to the maximum number of processors used. Overall, our grid splitting investigations show that a larger number of smaller blocks improves computational load balance and parallel scalability; however, there is a tradeoff since a large number of splits adversely affects efficiency due to an increase in the surface-to-volume ratio of grid points.

4.3 Grid Grouping Results

We compare our EVAH-based heuristic grid grouping strategy with naive bin-packing only for Case 2, and investigate their role on the quality of load balancing. All timing results are presented in Table 3 (the performance data for bin-packing is reproduced from Table 2 for easier comparison).

Table 3: Runtimes (in seconds) and load imbalance factors with bin-packing and EVAH heuristic grouping strategies, and asynchronous communication for Case 2

	Bin-packing						
P	T_{exec}	T_{comp}^{max}	T_{comp}^{avg}	T_{comm}^{max}	T_{comm}^{avg}	LB_{comp}	LB_{comm}
32	32.0	29.3	23.0	1.30	0.90	1.27	1.44
64	13.5	11.9	10.8	0.67	0.55	1.10	1.22
128	7.6	6.2	5.4	0.60	0.52	1.15	1.15
256	5.5	3.7	2.8	0.88	0.50	1.32	1.76
320	4.7	2.9	2.2	0.57	0.46	1.32	1.24
384	4.7	2.9	1.9	0.99	0.56	1.53	1.77
448	4.5	3.0	1.7	0.85	0.46	1.76	1.85
	EVAH heuristic						
P	T_{exec}	T_{comp}^{max}	T_{comp}^{avg}	T_{comm}^{max}	T_{comm}^{avg}	LB_{comp}	LB_{comm}
32	26.2	23.3	21.8	1.50	1.28	1.07	1.17
64	13.0	11.3	10.8	0.76	0.66	1.05	1.15
128	7.3	5.8	5.4	0.99	0.89	1.07	1.11
256	4.8	3.2	2.7	0.77	0.67	1.19	1.15
320	4.4	3.0	2.3	0.76	0.65	1.30	1.17
384	4.7	3.1	1.9	0.97	0.55	1.63	1.76
448	4.3	3.0	1.7	0.73	0.64	1.76	1.14

The results in Section 4.2 showed that the quality of communication load balancing LB_{comm} is a big drawback of the bin-packing grouping strategy. Table 3 demonstrates that the EVAH heuristic technique improves this factor considerably. In fact, except for $P = 384$, LB_{comm} using EVAH is at most 1.17. However, T_{comm}^{avg} is always larger for EVAH, and is 5–15% of T_{exec} (compared to 3–12% for bin-packing). This is somewhat expected since the overall goal of the heuristic grouping strategy is to reduce T_{exec}.

Performance scalability for both strategies when using more than 256 processors is low due to our fixed problem size. For example, when $P = 484$, each group contains, on average, only two grids (since $Z = 857$). With such a low number of blocks per group, the effectiveness of any strategy is diminished; moreover, the communication overhead relative to computation may increase substantially. For instance, with low processor counts, the communication-to-computation ratio is less than 7%, but grows to more than 35% with higher counts.

5 Summary and Conclusions

The overset grid method is a powerful technique for high-fidelity CFD simulations about complex aerospace configurations. In this paper, we presented and analyzed three parallel performance enhancement techniques for efficient computations of such large-scale realistic applications on state-of-the-art supercomputers. Specifically, the role of asynchronous communication, grid splitting, and grid grouping strategies were discussed. The asynchronous exchange relaxed the communication schedule in order to hide latency. Grid splitting was used to improve computational load balance while retaining the same grid system on different numbers of processors. Finally, a heuristic grid clustering technique balanced interprocessor communication with the goal of reducing the overall execution time.

All experiments were performed with the OVERFLOW-D Navier-Stokes code on a 512-processor Origin2000 system at NASA Ames Research Center. The CFD problem was the simulation of vortex dynamics in the complex flow region for hovering rotors. The grid systems for our three test cases consisted between 454 and 1679 overset grids, and varied in size from 63 million to 78 million grid points. The asynchronous communication strategy reduced execution time by more than a factor of two by significantly reducing the communication overhead. Grid splitting improved the workload balance by increasing the number of grids; however, the relative communication cost was adversely affected due to a larger surface-to-volume ratio of grid points. The heuristic grid grouping strategy compared extremely favorably with the original bin-packing technique. It improved the communication balance considerably while reducing the execution time. Overall results indicated that all three performance enhancement techniques are very effective in improving the quality of load balance and reducing execution time for overset grid applications.

Further improvements in the scalability of the overset grid methodology could be sought by using a more sophisticated parallel programming paradigm especially when the number of blocks Z is comparable to the number of processors P, or even when $P > Z$. One potential strategy that can be exploited on SMP clusters is to use a hybrid MPI+OpenMP multilevel programming style [3]. This approach is currently under investigation.

Acknowledgements

The work of the first author was supported by NASA Ames Research Center under Contract Number DTTS59-99-D-00437/A61812D with AMTI/CSC. The authors would like to thank Prof. N. Lopez-Benitez at Texas Tech University for his help with EVAH.

References

[1] P.G. Buning, W. Chan, K.J. Renze, D.Sondak, I.T. Chiu, J.P. Slotnick, R. Gomez, and D. Jespersen, *Overflow User's Manual Version 1.6au,* NASA Ames Research Center, Moffett Field, CA, 1995.

[2] M.J. Djomehri, R. Biswas, R.F. Van der Wijngaart, and M. Yarrow, "Parallel and distributed computational fluid dynamics: Experimental results and challenges," in: *Proc. 7th Intl. High Performance Computing Conf.*, LNCS 1970 (2000) 183–193.

[3] M.J. Djomehri and H. Jin, "Hybrid MPI+OpenMP programming of an overset CFD solver and performance investigations," NASA Ames Research Center, NAS Technical Report NAS-02-002 (2002).

[4] N. Lopez-Benitez, M.J. Djomehri, and R. Biswas, "Task assignment heuristics for distributed CFD applications," in: *Proc. 30th Intl. Conf. on Parallel Processing Workshops*, (2001) 128–133.

[5] R. Meakin, "A new method for establishing inter-grid communication among systems of overset grids," in: *Proc. 10th AIAA Computational Fluid Dynamics Conf.*, Paper 91-1586 (1991) 662–676.

[6] R. Meakin, "On adaptive refi nement and overset structured grids," in: *Proc. 13th AIAA Computational Fluid Dynamics Conf.*, Paper 97-1858 (1997).

[7] R. Meakin and A.M. Wissink, "Unsteady aerodynamic simulation of static and moving bodies using scalable computers," in: *Proc. 14th AIAA Computational Fluid Dynamics Conf.*, Paper 99-3302 (1999).

[8] Y. Saad, *Iterative Methods for Sparse Linear Systems*, PWS Publishing Company, Boston, MA, 1996.

[9] J. Steger, F. Dougherty, and J. Benek, "A Chimera grid scheme," *ASME FED*, 5 (1983).

[10] R.C. Strawn and J.U. Ahmad, "Computational modeling of hovering rotors and wakes," in: *Proc. 38th AIAA Aerospace Sciences Mtg.*, Paper 2000-0110 (2000).

[11] R.C. Strawn and M.J. Djomehri, "Computational modeling of hovering rotor and wake aerodynamics," in: *Proc. 57th Annual Forum of the American Helicopter Society*, (2001).

[12] R. Van Driessche and D. Roose, "Load balancing computational fluid dynamics calculations on unstructured grids," in: *Parallel Computing in CFD*, AGARD-R-807 (1995) 2.1–2.26.

[13] A.M. Wissink and R. Meakin, "Computational fluid dynamics with adaptive overset grids on parallel and distributed computer platforms," in: *Proc. Intl. Conf. on Parallel and Distributed Processing Techniques and Applications*, (1998) 1628–1634.